LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-463573

Silo & HDF5 I/O Scaling Improvements on BG/P Systems

M. R. Collette, M. C. Miller

December 3, 2010

NECDC 2010
Los Alamos, NM, United States
October 18, 2010 through October 22, 2010

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Silo & HDF5 I/O Scaling Improvements on BG/P Systems

M.R. Collette, M.C. Miller;**

**Lawrence Livermore National Laboratory, Livermore, California, 94550;*

Silo and HDF5 are I/O libraries used by many codes important to the LLNL's Weapons and Complex Integration (WCI) mission. In the past year, modest adjustments and tuning of Silo, HDF5 and the I/O configuration of our BG/P platform, Dawn, were undertaken. A key goal of this work was to improve I/O performance without requiring any changes in the application codes themselves. In particular, the application codes have been allowed to continue to use a simplified yet highly flexible I/O paradigm known as "Poor Man's Parallel I/O", where scalability is achieved through concurrent, serial I/O to multiple files. Our results demonstrate substantial performance gains (better than 50x in many cases) at large scale (greater than 64,000 MPI tasks). We describe key enhancements made to Silo, HDF5 and the I/O configuration of our BG/P platform and present very favorable results from scalability studies over a wide range of operating scenarios.

Scaling & I/O on BG/P systems

LLNL's Dawn machine is the 500-teraFLOPS Advanced Simulation and Computing (ASC) program's Sequoia Initial Delivery System of the same lineage as LLNL's BlueGene/L system. Dawn is an IBM BlueGene/P (BG/P) architecture with 36,864 quad-code 850MHz PowerPC 450 processors, yielding 147,456 total CPUs. After porting a large scale ASC 2D & 3D multi-physics code onto it, we performed various scaling & I/O studies up to the full system and analyzed the results.

Unexpectedly Poor I/O Performance

Our I/O performance on previous systems has been quite acceptable and has scaled well when used with parallel file systems such as Lustre. However, on Dawn, our file writing times were tens of minutes and our read times measured in hours, for I/O operations performed with just 1024 processors that completed in less than 30 seconds on all our previous systems. Even worse, these I/O times increased as we scaled to use more processor.

Investigation And Analysis

Our code employs the silo and HDF5 I/O libraries to read and write its restart data files. Silo utilizes the HDF5 file format and can implement a variety of I/O drivers that control low-level I/O operation details such as the stdio driver, and sec2 driver which is our default. We performed many scaling studies and tested a variety of different established drivers and determined that employing the 'core' I/O driver, where all the data is kept in memory until the file is closed and written to disk all at once, showed a huge improvement, 50 times speedup for writes and a 100 times speedup for reads.

Less Than Ideal Solution

However, this core driver requires there to be enough memory free to hold the entire restart data file, which could be hundreds of megabytes and a significant percentage of all available memory, particularly on somewhat low memory machines like Dawn where each processor might only have 1GB.

Plus, the scaling performance of this solution still resulted in multiple hours for read times when using most of the machine. A better solution had to be devised.

Uncovering The Root Cause

It was speculated that metadata, the tiny informational bits that describe data being

written, such as its type, structure, or even hierarchy within the file (essentially data about the data) inherently written along with all our data (be that large or small) was crippling the performance when written by itself to disk, which we confirmed was happening. It was also noted that HDF5 itself seemed to have inefficiencies with having its file pointer jump around rather unnecessarily when writing various portions of data, increasing seek times.

A Better Solution Was Implemented

A new I/O driver was devised and written that was essentially a hybrid of the core and default drivers, allowing buffering & packing of the tiny troublesome metadata while not taking up much memory with large data chunks by efficiently writing them to disk immediately, along with other improvements. Also a variable limit was incorporated for the amount of memory to use as a file data buffer which can be optimized to conform to the most efficient I/O block size for the file system.

Environment Settings Also A Factor

It was also noted that Dawn was our first platform with a different default setting for its DirectIO flag, which controls data buffering on the file system side of I/O. The default was changed so that large chunks of data were more optimally written to disk. However, this degraded small I/O write performance and exasperated our troublesome metadata performance issues. We confirmed this and performed scaling studies with this DirectIO flag changed which demonstrated that while our write performance was not particularly sensitive to this setting, our read performance improved by at least a factor of four at small scale, and beyond a factor of 20 at large scale. It was also determined that, quite unfortunately, this flag was not modifiable at the code level nor could be disabled at the file level under Dawn's lightweight kernel limitations unlike how it may be modified on most other systems. Shortly after alerting the Livermore Computing center of these results, the default value of this DirectIO flag was changed.

Other Improvements

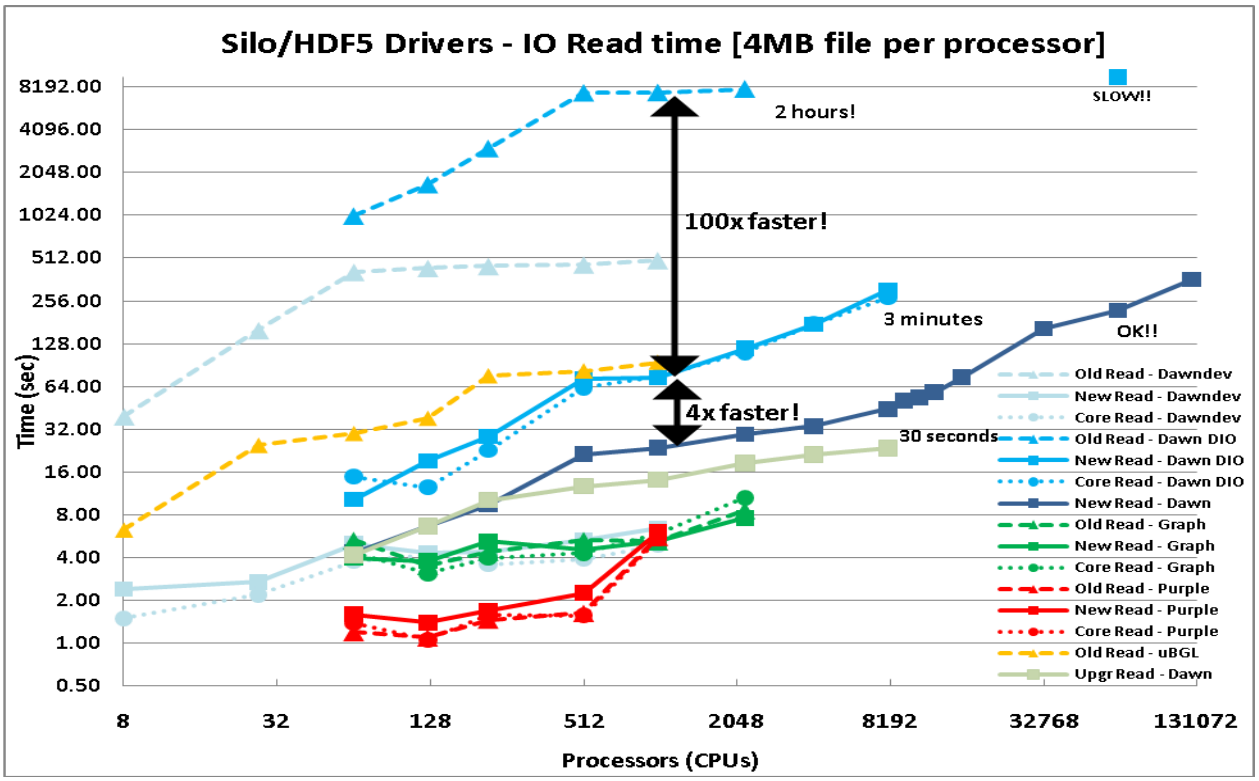
We also analyzed in detail the order in which our data was being read in from disk and reordered our data writes to logically conform to that, which matched up the order of reads and writes to streamline the read process and keep the file pointer from jumping around and thus reduce seek times. For example, we would often write the length of arrays to the file after the array itself was written, but when reading in an array you must first know its length in order to allocate enough memory for it. Writing the length first keeps the system from hunting thru the file for the necessary data since it is just the next bit of info it encounters.

Results Of Our Efforts

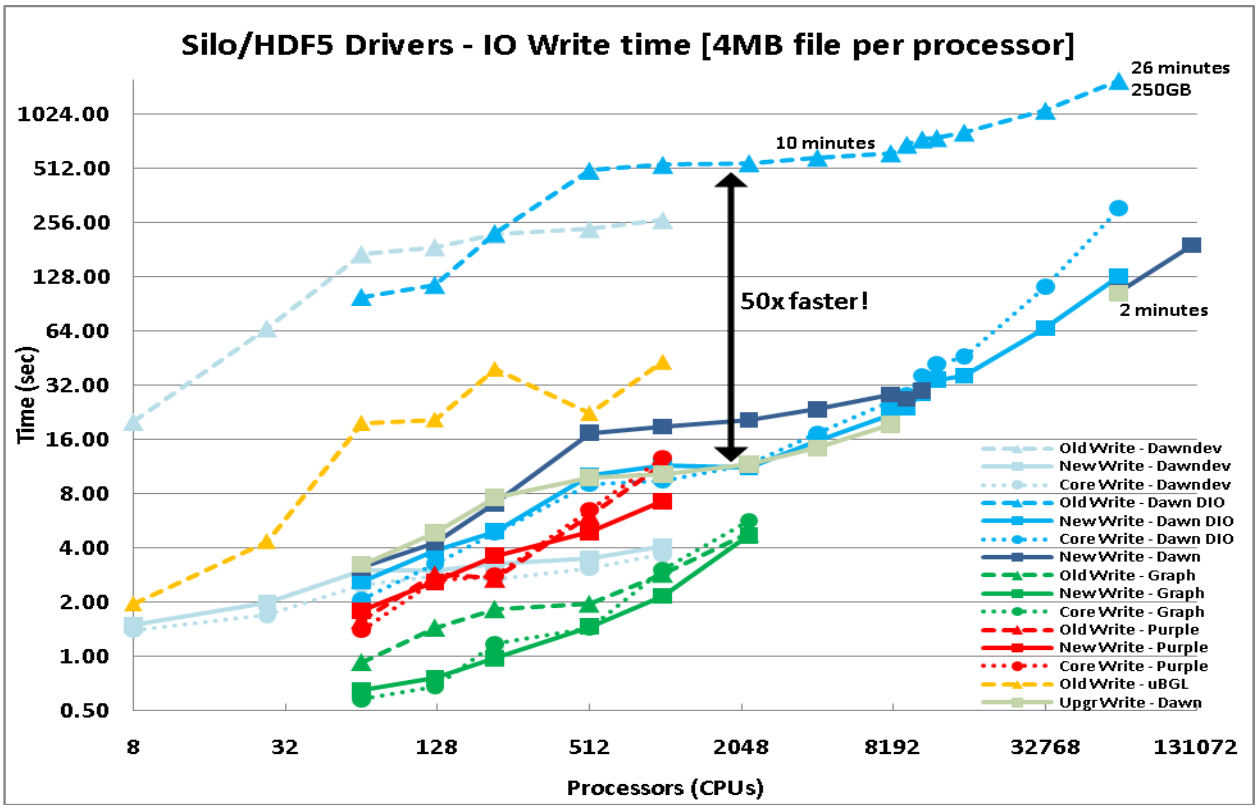
In the end, with all our improvements, our test problems exhibit a factor of 50 improvement in write times, and over 400 times faster read times. Even as our I/O measurements scaled to using most of the machine, our performance was on the order of a few minutes instead of many hours. We are pleased with this improved performance. Our read performance is even faster than when using the 'core' driver, which kept the whole data file in memory, yet now we don't have to reserve any such huge chunk of our limited memory for such a purpose, and our scaling performance has been greatly improved as well. (Graphs 1 - 4).

NECDC '10 Poster: LLNL-POST-456996

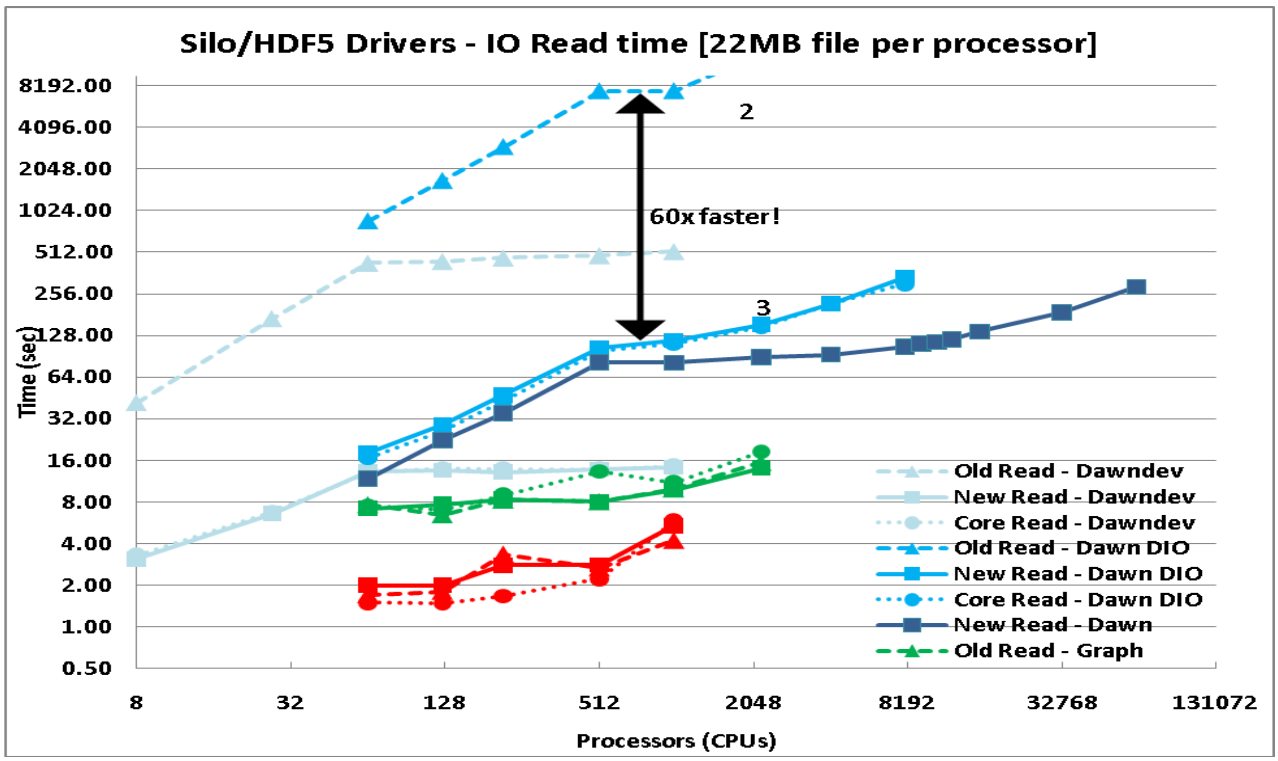
This work was performed under auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA2734



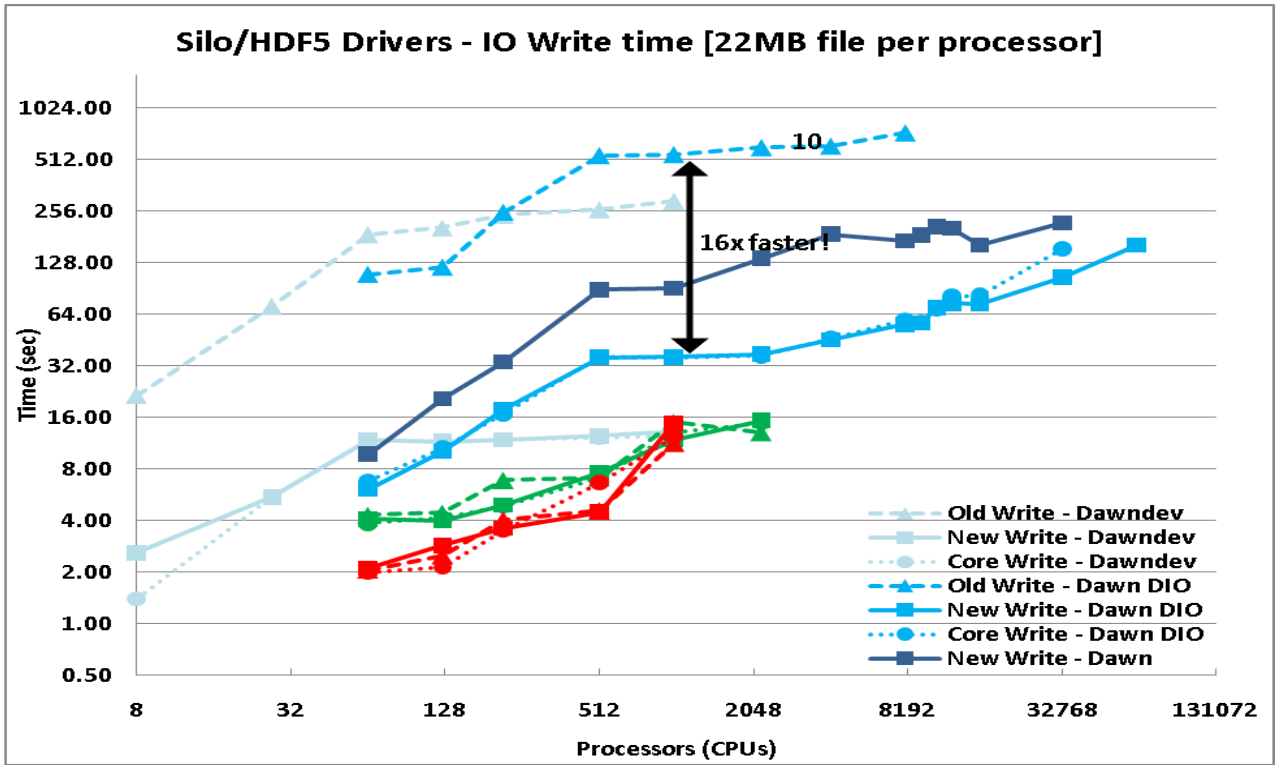
Graph 1. Scaling Read Times By Driver And Machine



Graph 2. Scaling Write Times By Driver And Machine



Graph 3. Scaling Larger Read Times By Driver And Machine



Graph 4. Scaling Larger Write Times By Driver And Machine